Scalable microbiome metagenome assembly and profiling

Sébastien Boisvert
Laval University, Canada

Talk given at
Argonne National Laboratory, Illinois, USA

Tuesday 2012-10-16 1:30pm-2:30pm

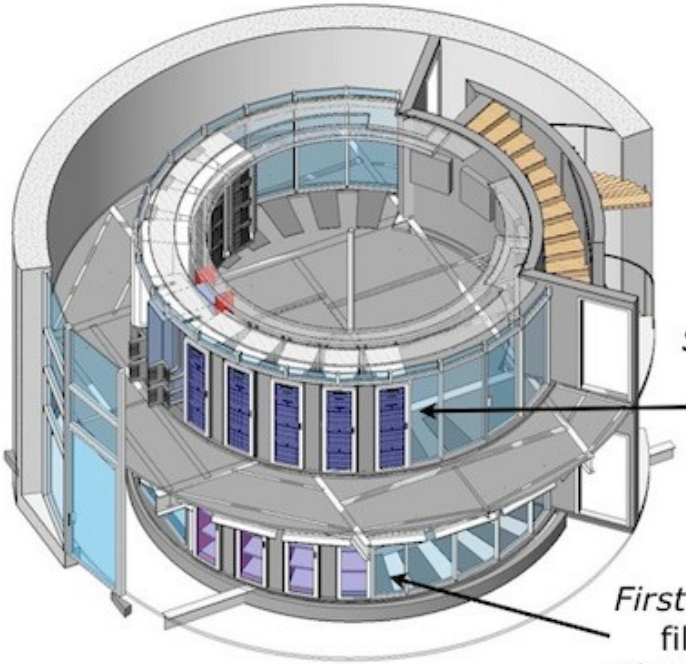
Invited by Professor Rick Stevens

http://www.mcs.anl.gov/events/detail.php?id=1857
TCS Bldg. 240, Room 4301, Argonne National Laboratory
Host: Fangfang Xia

UNIVERSITÉ LAVAL

Where is Laval University ?

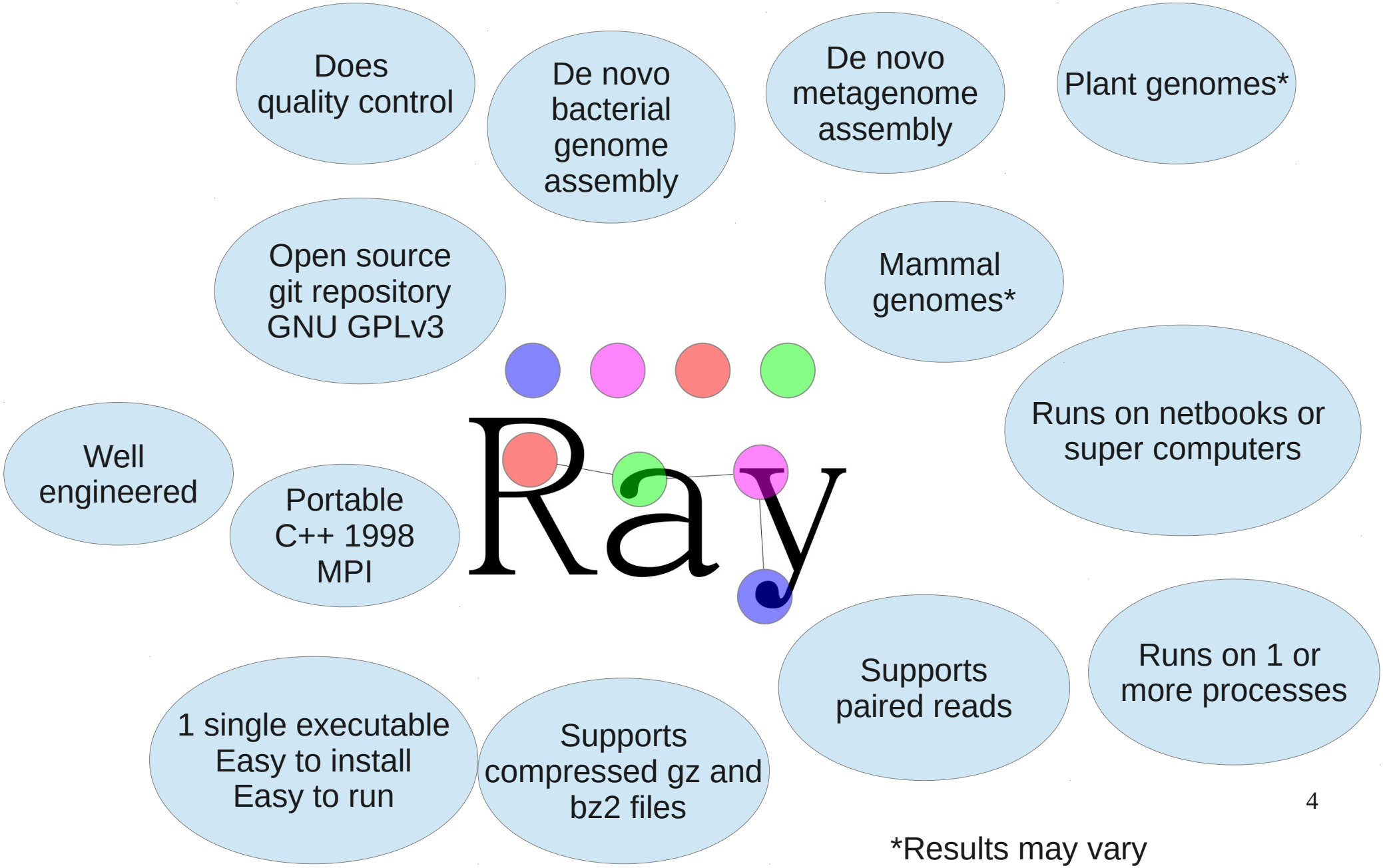
Super computing at Laval University



colosse
#314 top500 06/2012
7616 Intel Xeon X5560 cores
Mellanox Technologies MT26428
332 kW

2010 InfoWorld GREEN 15

Why care about Ray?



*Results may vary

Plan

- Problematic
- Some computer science
- RayPlatform (framework and engine to create distributed software tools)
- Ray (base plugins for de novo assembly)
- Ray Meta (de novo metagenome assembly)
- Ray Communities (biological profiling)

Parallel DNA sequencers

- Read DNA in parallel
- Many relatively short reads
- 6 000 000 000 reads / 11 days on 1 HiSeq 2000
- The HiSeq: “GPU” of biotechnologies
- Distributed file systems accommodate the volume
  - Lustre, IBM GPFS, Ceph
- Linux VFS accommodate access to data
- Distributed scalable analysis tools are lacking

Metagenomics

- Many bacterial genomes
- Huge samples in earth sciences (environmental samples)
- Needs to scale beyond the 1-process model
- de novo metagenome assemblers:
  - Genovo, MetaVelvet, Meta-IDBA, Kiki, Ray Meta
- Only Kiki and Ray Meta are distributed
- MetaVelvet (Namiki et al. 2012 NAR) and Meta-IDBA (Peng et al. 2011 Bioinformatics) partition by connected components and coverage frequencies
- Genovo (Laserson et al. 2011 JCB) was used on 311k reads

Parallel programming models

- 1 process with many kernel threads on 1 machine
- Many processes with IPC (interprocess communication)
- Many processes with MPI (message passing interface)

Message passing with MPI

- MPI 3.0 contains a lot of things
- Point-to-point communication (two-sided)
- RDMA (one-sided communication)
- Collectives
- MPI I/O
- Custom communicators
- Many other features

Point-to-point versus collectives

- With point-to-point, the dialogue is local between two folks
- Collectives are like meetings – not productive when too many of them
- Collectives are not scalable
- Point-to-point is scalable

MPI is low level

- Message passing does not structure a program
- Needs a framework
- Should be modular
- Should be easy to extend
- Should be easy to learn and understand

Models for message passing

- 2 kernel threads per process (1 for busy waiting for communication and 1 for processing)
- Cons:
  - not lock-free
  - prone to programming errors
  - Half of the cores busy wait (unless they sleep)

Granularity

- Standard sum from 1 to 1000
- Granular version: sum 1 to 10 on the first call, 11 to 20 on the second, and so on
- Many calls are required to complete

Models for message passing

- 1 single kernel thread per process
- Comm. and processing interleaved
- Con:
  - Needs granular code everywhere !
- Pros
  - Efficient
  - Lock-free (less bugs)

Models for task splitting

- **Model 1: separated duties**
- Some processes are data stores (80%)
- Some processes are algorithm runners (20%)
- Con:
  - Data store processes do nothing when nobody speak to them
  - Possibly unbalanced

Models for task splitting

- **Model 2: everybody is the same**
- Every process has the same job to do
- But with different data
- One of the processes is also a manager (usually # 0)
- Pros
  - Balanced
  - All the cores work equally

Memory models

- 1. Standard: 1 local virtual address space per process
- 2. Global arrays (distributed address space)
  - Pointer dereference can generate a payload on the network
- 3. Actors, each process has its own stuff
  - Message passing
  - DHTs (distributed hash tables)
  - DHTs are nice because the distribution is uniform

Choosing the programming language

- Needs something portable
- Needs something fast
- Needs objects (private stuff, design patterns)
- Needs pointers
- Needs function pointers (or any callback method)
- Needs MPI bindings
- C++ (ISO/IEC 14882:1998)

State machine

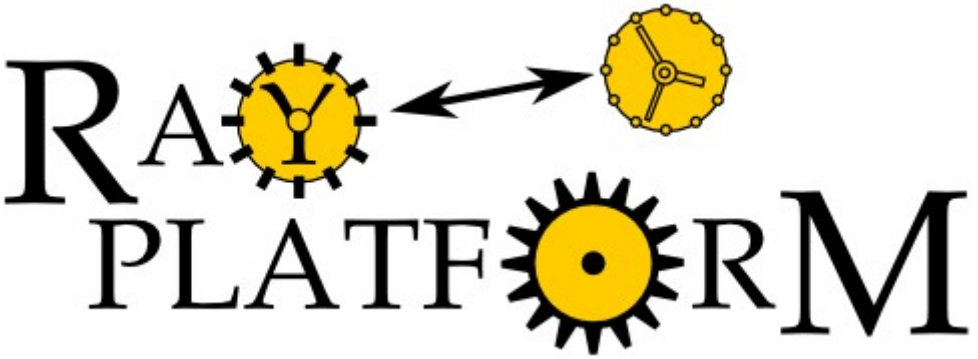
- A machine with states
- Behaviour guided by its states
- Each process is a state machine

Definitions

- Handle: opaque label
- Handler: behaviour associated to an event
- Plugin: orthogonal module of the software
- Adapter: binds two things that can not know each other
- Core: the kernel
- Handler table: tells which handler to use with any handle
- Handler table is like interruption table

RayPlatform

- Each process has: inbox, outbox
- Only point-to-point
- Modular plugin architecture
- Each process is a state machine
- The core allocates:
  - Message tag handles
  - Slave mode handles
  - Master mode handles
- Associate behaviour to these handles
- GNU Lesser General Public License, version 3
- https://github.com/sebhtml/RayPlatform



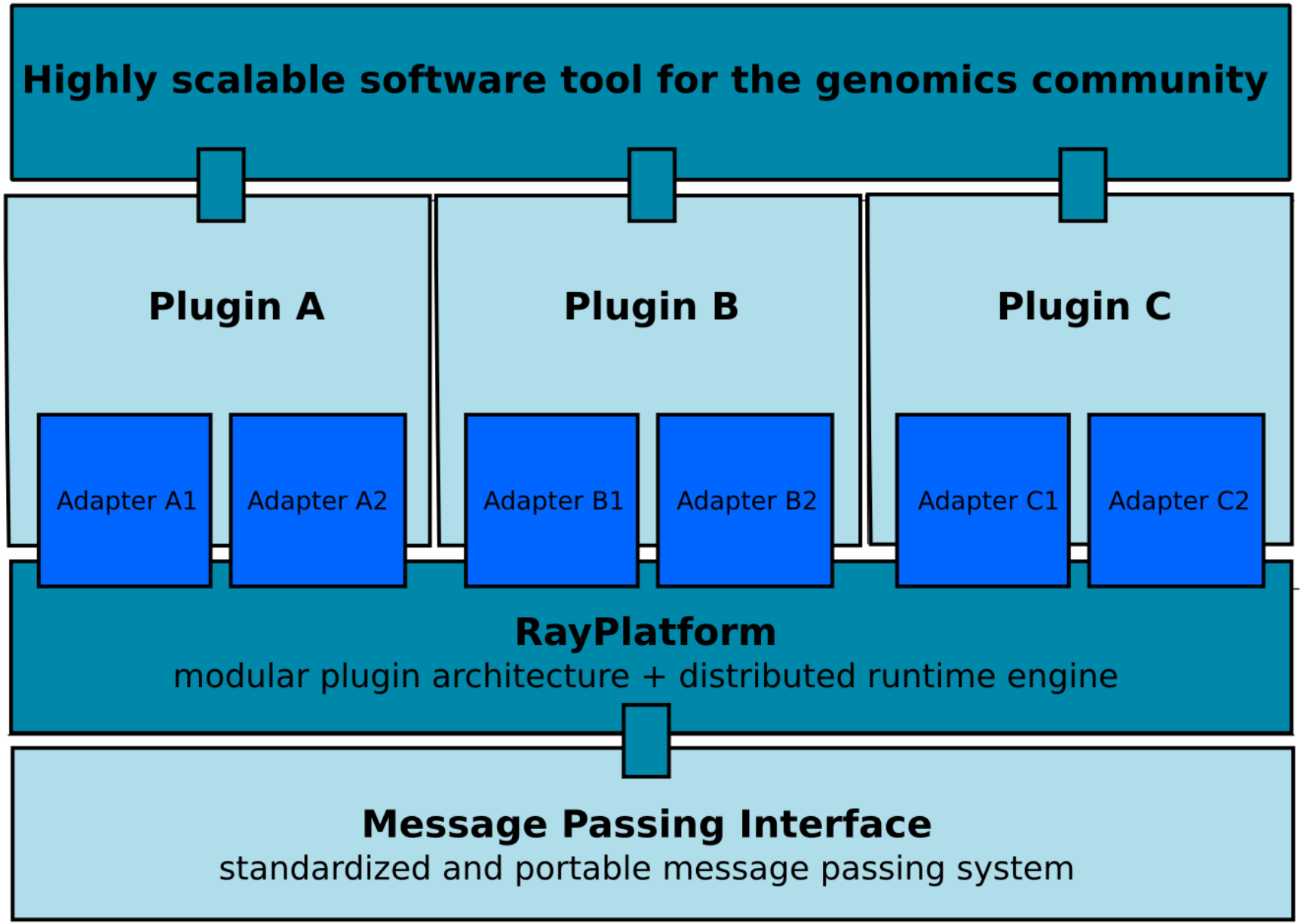
Main loop

- while(isAlive()){

    receiveMessages();
    processMessages();

    processData();
    sendMessages();

  }



Quick facts:

- RayPlatform assists developers during the content creation
- A software tool for the genomics community is implemented as plugins
- Plugins are executed by a distributed runtime engine
- Communication is portable thanks to MPI

Some counts for RayPlatform

- 8637 lines of code
- 4185 comment lines
- 3920 blank lines
- 53 C++ classes

Virtual processor (VP)

- Problem: kernel threads have a overhead, but it is nice to have threads
- Solution: each process has many user space threads (workers) that push messages
- The operating system is not aware of workers (user space threads)

Virtual communicator (VC)

- Problem: sending many small messages is costly
- Solution: aggregate them transparently
- Workers push messages on the VC
- The VC pushes bigger messages in the outbox
- Workers are user space threads
- States: Runnable, Waiting, Completed

Virtual message router

- Problem: any-to-any communication pattern can be bad
- Solution: fit the pattern on a better graph
- 5184 processes -> 26873856 comm. edges ! (diameter: 1)
- With surface of regular convex polytope: 5184 vertices, 736128 edges, degree: 142, diameter: 2

Hash tables in RayPlatform

- MyHashTable.h, MyHashTableGroup.h
- C++ template
- Sparse (Knuth model, 64 buckets / group)
- Distributed (DHT)
- Open addressing (double hashing)
- Double hashing has no clustering
- But is bad with CPU cache
- Incremental resizing

Hardware acceleration for hash table

- Needs to count the bits to 1 in bitmaps
- Pop count does that
- if available: popcnt (proposed by David Weese)

Profiling is understanding

- RayPlatform has its own real-time profiler
- Reports messages sent/received, current slave mode at every 100 ms quantum

Example

- Rank 0: RAY_SLAVE_MODE_ADD_VERTICES Time= 4.38 s Speed= 74882 Sent= 51 (**processMessages**: 28, **processData**: 23) Received= 52 Balance= -1

  Rank 0 received in **receiveMessages**:

  Rank 0 RAY_MPI_TAG_VERTICES_DATA 28

  Rank 0 RAY_MPI_TAG_VERTICES_DATA_REPLY 24

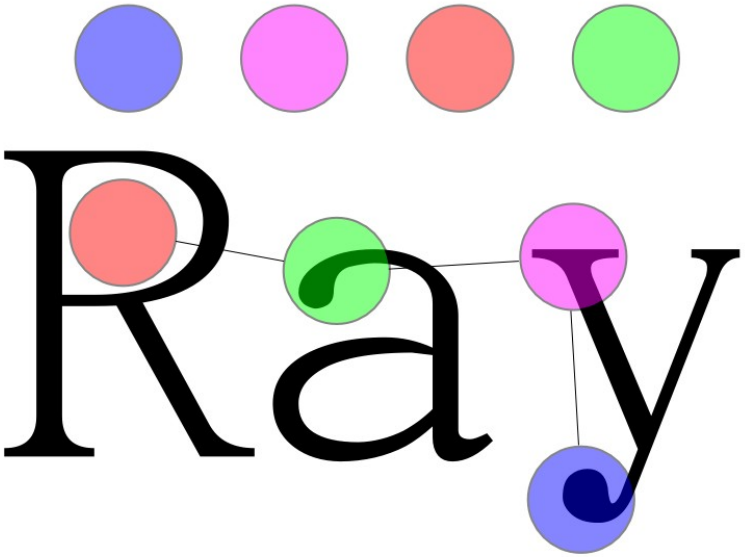
  Rank 0 sent in **processMessages**:

  Rank 0 RAY_MPI_TAG_VERTICES_DATA_REPLY 28

  Rank 0 sent in **processData**:

  Rank 0 RAY_MPI_TAG_VERTICES_DATA 23

Ray



- Built on top of RayPlatform
- A set of plugins
- Plugins implement solutions
  - Ray Assembler (Boisvert et al. 2010)
  - Ray Meta (in minor revision)
  - Ray Communities
- GNU General Public License, version 3
- https://github.com/sebhtml/ray

Some counts for Ray

- 22 plugins
- 31607 lines of code
- 6513 comment lines
- 11326 blank lines
- 88 C++ classes
- 36 master modes
- 30 slave modes
- 122 message tags

de Bruijn graphs in bioinformatics

- Alphabet: {A,T,C,G}, word length: k
- Vertices V = {A,T,C,G}^k
- Edges are a subset of V x V
- (u,v) is an edge if the last k-1 symbols of u are the first k-1 symbols of v
- Exemple: A**TCGA** -> **TCGA**T
- In genomics, we use a de Bruijn subgraph using k-mers for vertices and (k+1)-mers for edges
- k-mers and (k+1)-mers are sampled from data
- Idury & Waterman 1995 Journal of Computational Biology

Distributed storage engine

- Reads are distributed uniformily
- K-mers are distributed uniformily
- Only 1 of any 2 reverse complement k-mers stored
- Annotations on objects (be it reads or k-mers)
- Virtual coloring of k-mers
- Compact edge representation (Simpson et al. 2009 Genome Research)

Sequencing errors

- Bloom filter, 2 operations: hasItem?, insertItem!
- No false negatives, few false positives
- In bioinformatics (Pell et al. PNAS 2012)
- Each Ray process has a Bloom filter
- Weeds out most of the k-mers occurring once

Broad utility of Ray

- Runs on 1 or more processes (from netbooks to super computers)
- Processes can be on the same machine or not
- Bacterial genome assembly (Boisvert et al. 2010 Journal of Comp. Biol.)
- Large genome assembly (snake, bird, fish, Assemblathon 2)
- Plant genomes
- Metagenome assembly and profiling (Boisvert et al. in revision)
- Quality control (paired end distribution, coverage distribution)
- Profiling with colored de Bruijn graph (Iqbal et al. 2012 Nature Genetics)

Assembly with graph traversal

- Heuristic-guided traversal (Jackson et al. 2010 IEEE International Parallel & Distributed Processing Symposium (IPDPS))
- Huge distributed graph (Satish et al. SC2012 International Conference for High Performance Computing, Networking, Storage and Analysis)
- Contigs are paths in the graph (Pevzner et al. 2001 PNAS)

Assembly with graph traversal (2)

- Each process has a set of assembly seeds
- Seeds are similar to unitigs (Boisvert et al. Journal of Computational Biology)
- Any seed will be extended to “grow” a contig from a “seed”

Image: Wikipedia

Some results with Ray Meta

- All these results are on Colosse
- Round-trip in-application point-to-point latency > 100 microseconds for 512-process jobs
- 3 000 000 000 reads from a 1000-bacterium metagenome, 15 hours on 1024 cores
- 400 000 000 reads from 100-bacterium metagenome, 14 hours, 128 cores
- Includes also k-mer based profiling (genome abundance, taxonomy, gene ontology)

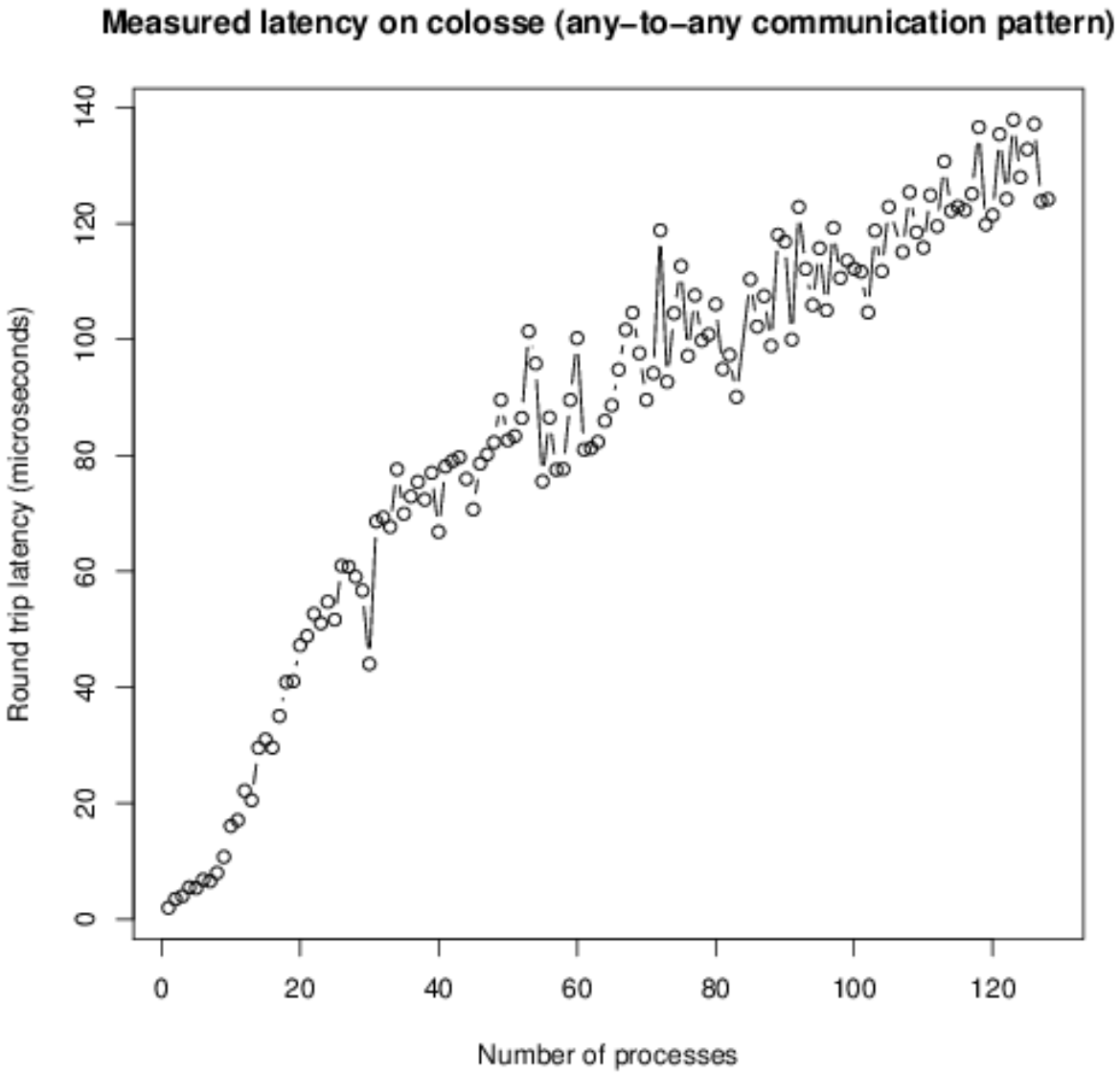
Steps for 1000-genome

- Network testing: 3 minutes, 55 seconds
- Counting sequences to assemble: 2 minutes, 12 seconds
- Sequence loading: 24 minutes, 32 seconds
- K-mer counting: 32 minutes, 50 seconds
- Coverage distribution analysis: 3 seconds
- Graph construction: 1 hours, 21 minutes, 35 seconds
- Null edge purging: 28 minutes, 3 seconds
- Selection of optimal read markers: 44 minutes, 11 seconds
- Detection of assembly seeds: 46 minutes, 58 seconds
- Estimation of outer distances for paired reads: 23 minutes, 36 seconds
- Bidirectional extension of seeds: 3 hours, 25 minutes, 50 seconds
- Merging of redundant paths: 4 hours, 27 minutes, 55 seconds
- Generation of contigs: 5 minutes, 48 seconds
- Scaffolding of contigs: 2 hours, 4 minutes, 7 seconds
- Counting sequences to search: 19 seconds
- Graph coloring: 18 minutes, 18 seconds
- Counting contig biological abundances: 3 minutes, 44 seconds
- Counting sequence biological abundances: 31 minutes, 50 seconds
- Loading taxons: 22 seconds
- Loading tree: 14 seconds
- Processing gene ontologies: 6 seconds
- Computing neighbourhoods: 0 seconds
- Total: 15 hours, 46 minutes, 41 seconds

Latency matters

- To build the graph for the dataset SRA000271 (human genome, 4 Giga reads), with 512 processes
  - 159 min when average latency is 65 us
  - 342 min when average latency is 260 us
- 4096 processing elements, Cray XE6, round-trip latency in application -> 20-30 microseconds (Carlos Sosa, Cray Inc.)

Latency with Mellanox Technologies MT26428 (Colosse)

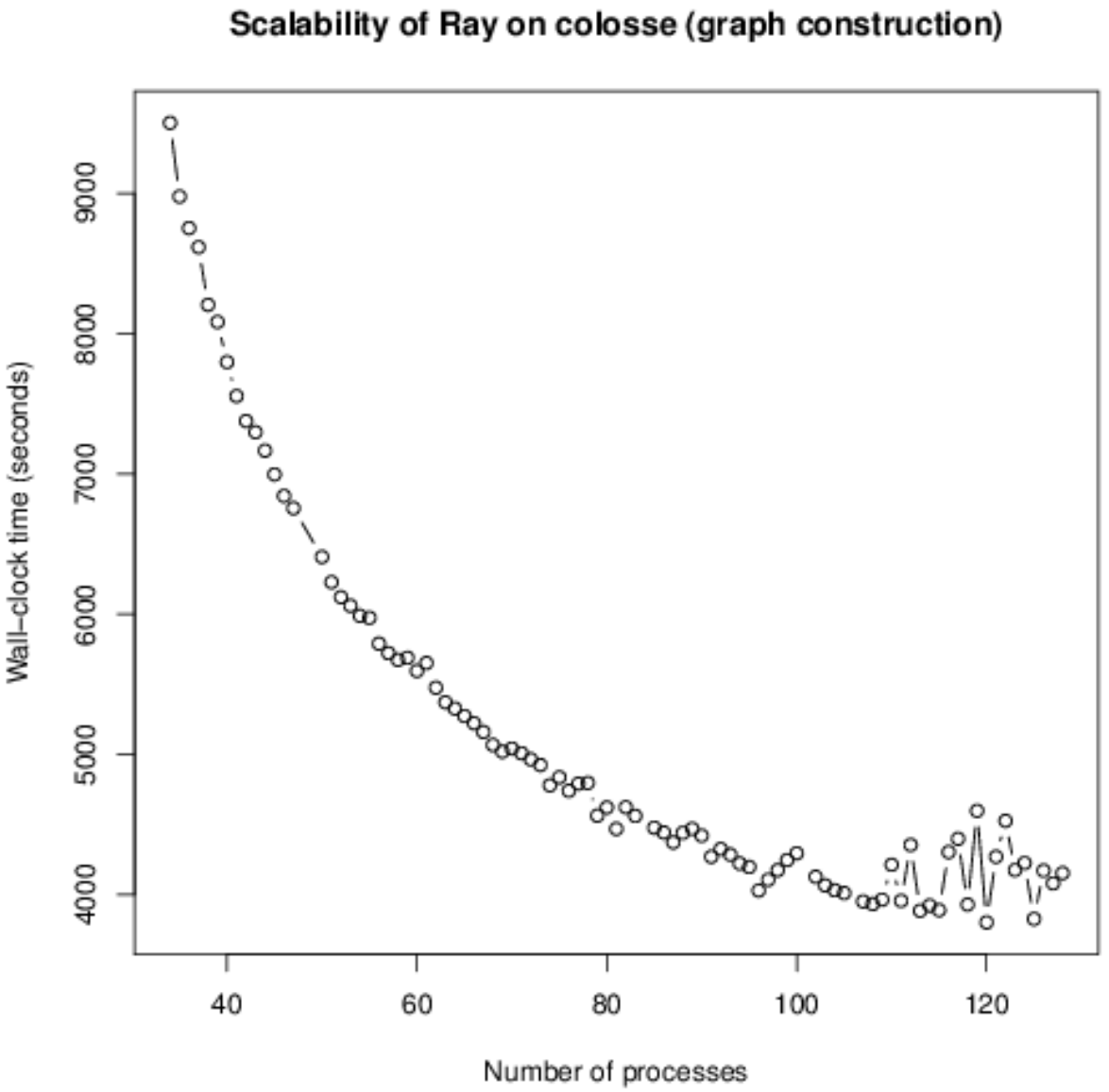


IBM iDataPlex

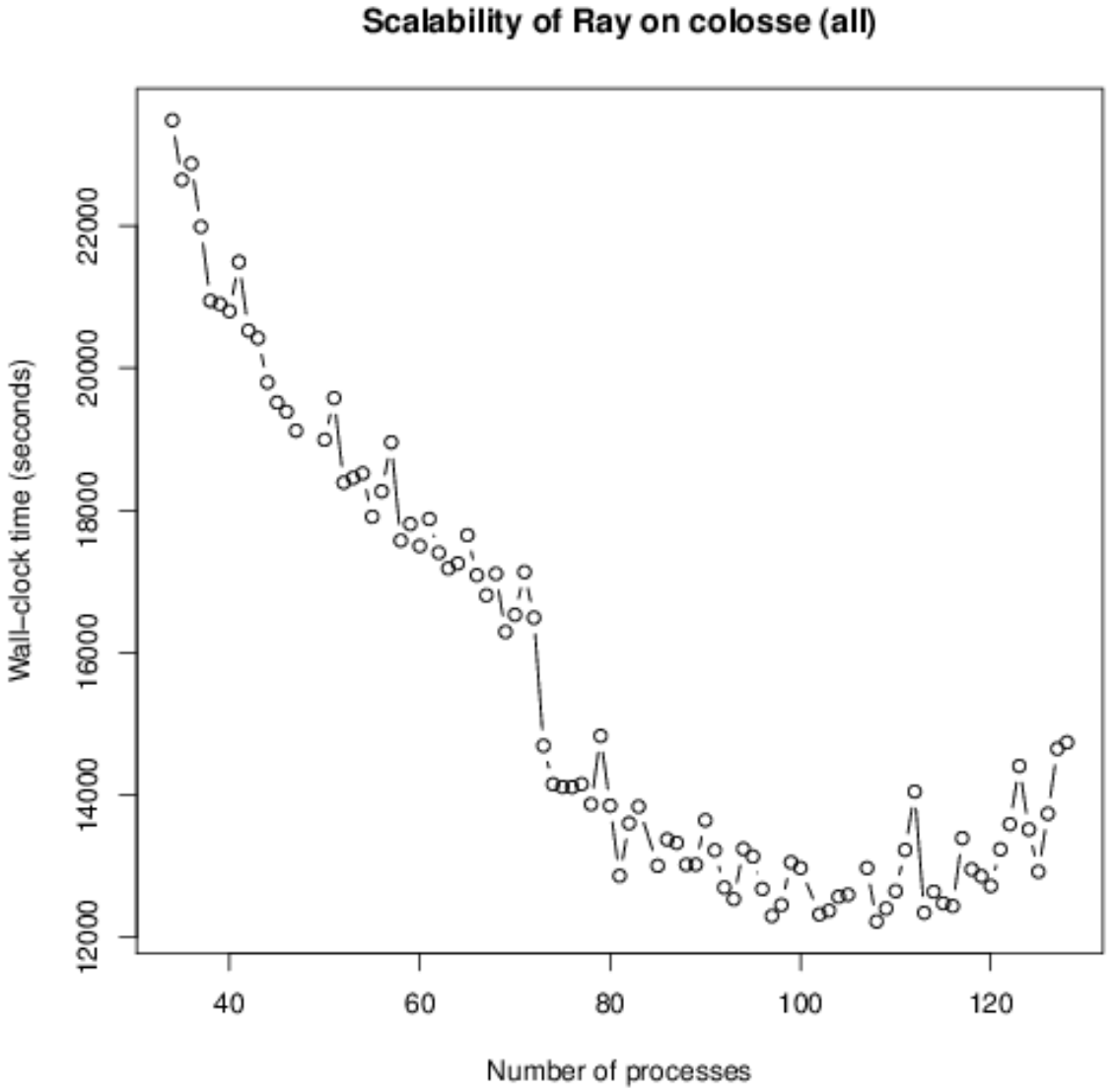
- Latency is constant on 2 tested IBM iDataPlex, on Cray XE6, on large SMP machines
- Latency increases linearly on some machines with Mellanox Technologies MT26428

Building the distributed de Bruijn graph

- metagenome
- sample SRS011098
- 202 Mega reads



Overall (SRS011098)



Ray Communities

- Add colors to k-mers
- Similar to Iqbal et al. 2012 Nature Genetics
- No coverage channels
- Virtual coloring – many k-mers will reference exactly the same physical colors
- Manuscript about Ray Meta and Ray Communities in revision

Development

- Done on a 128 GB 32-core SMP machine (ssh from netbook)
- Tools: vim, git, screen, ssh, gcc, llvm, MPICH2, “Other”-MPI, PBS

Acknowledgements / Invitation

- Prof. Rick Stevens
- Host: Dr. Fangfang Xia
- Carolyn Peters (arrangements)

Acknowledgements / Funding

- SB: Doctoral award, **Canadian Institutes for Health Research**, September 2010 – August 2013
- Jacques Corbeil: **Canada Research Chair in Medical Genomics**
- Discovery Grants Program (Individual, Team and Subatomic Physics Project) from the **Natural Sciences and Engineering Research Council of Canada** (grant 262067 to François Laviolette)

Acknowledgements / Product team

- Sébastien Boisvert (designer, developer, release technician, community manager)
- Élénie Godzaridis (parallel designs, works in the industry)
- Prof. François Laviolette (graph specialist)
- Prof. Jacques Corbeil (genomician)
- Maxime Boisvert (design tricks, consultant in the industry)
- Dr. Frédéric Raymond (end user / stakeholder)
- Pier-Luc Plante (intern)

Acknowledgements / CPU time

- 2011: 50 core-years on Colosse
- 2012: 250 core-years on Colosse
- Compute Canada (Colosse, Mammouth Parallèle II, Guillimin)
- Calcul Québec, CLUMEQ, RQCHP
- Canadian Foundation for innovation for the 32-core 128-GB SMP machine
- Collaboration with Cray Inc. for the Cray XE6 (with Carlos Sosa)

Questions